Today’s Agenda

• Mutual Recursion
• Module System Example
  • Namespace Organization
  • Preserving Invariants
• Practice with Currying and High Order Functions
Mutual Recursion

• What if we need function f to call g, and function g to call f?
• This is a common idiom

```
fun earlier x =
  ...
  later x
  ...
fun later x =
  ...
  earlier x
  ...
```

Unfortunately this does not work 😞
Mutual Recursion Workaround

• We can use higher order functions to get this working
• It works, but there has got to be a better way!

```plaintext
fun earlier (f, x) =
  ...
  f x
  ...
fun later x =
  ...
  earlier (later, x)
  ...
```
Mutual Recursion with \textit{and}

- SML has a keyword for that
- Works with mutually recursive \texttt{datatype} bindings too

\begin{verbatim}
fun earlier x =
\ldots
later x
\ldots
\par \par \textbf{and} later x =
\ldots
earlier x
\ldots
\end{verbatim}
Module System

• Good for organizing code, and managing namespaces (useful, relevant)
• Good for maintaining invariants (interesting)
Currying and High Order Functions

- List.map!
- List.filter!
- List.foldl!
- Emacs unite!